

**Exception Handling in Java**

The **Exception Handling in Java** is one of the powerful mechanism to handle the runtime errors so that normal flow of the application can be maintained.

### Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. An exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

statement 1;

statement 2;

statement 3;

statement 4;

statement 5;//exception occurs

statement 6;

statement 7;

statement 8;

statement 9;

statement 10;

Suppose there are 10 statements in your program and there occurs an exception at statement 5, the rest of the code will not be executed i.e. statement 6 to 10 will not be executed. If we perform exception handling, the rest of the statement will be executed. That is why we use exception handling in Java.

## Java Exception Keywords

There are 5 keywords which are used in handling exceptions in Java.

|  |  |
| --- | --- |
| **Keyword** | **Description** |
| try | The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature. |

## Java Exception Handling Example

Let's see an example of Java Exception Handling where we using a try-catch statement to handle the exception.

**public** **class** JavaExceptionExample{

**public** **static** **void** main(String args[]){

**try**{

      //code that may raise exception

**int** data=100/0;

   }**catch**(ArithmeticException e)

{

System.out.println(e);

}

   //rest code of the program

   System.out.println("rest of the code...");

  }

}

**Output:**

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

## Common Scenarios of Java Exceptions

There are given some scenarios where unchecked exceptions may occur. They are as follows:

### A scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

**int** a=50/0;//ArithmeticException

### A scenario where NullPointerException occurs

If we have a null value in any variable, performing any operation on the variable throws a NullPointerException.

String s=**null**;

System.out.println(s.length());//NullPointerException

### A scenario where NumberFormatException occurs

The wrong formatting of any value may occur NumberFormatException. Suppose I have a string variable that has characters, converting this variable into digit will occur NumberFormatException.

String s="abc";

**int** i=Integer.parseInt(s);//NumberFormatException

### A scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result in ArrayIndexOutOfBoundsException as shown below:

**int** a[]=**new** **int**[5];

a[10]=50; //ArrayIndexOutOfBoundsException

**FileNotFound Exception**

|  |
| --- |
| //Java program to demonstrate FileNotFoundException  import java.io.File;  import java.io.FileNotFoundException;  import java.io.FileReader;   class File\_notFound\_Demo {        public static void main(String args[])  {          try {              // Following file does not exist              File file = new File("[E://file.txt](file:///E:\file.txt)");                FileReader fr = new FileReader(file);          } catch (FileNotFoundException e) {             System.out.println("File does not exist");          }      }  } |

**Output:**

File does not exist

**StringIndexOutOfBound Exception**

|  |
| --- |
| // Java program to demonstrate StringIndexOutOfBoundsException  class StringIndexOutOfBound\_Demo  {      public static void main(String args[])      {          try {              String a = "This is like chipping "; // length is 22              char c = a.charAt(24); // accessing 25th element              System.out.println(c);          }          catch(StringIndexOutOfBoundsException e) {              System.out.println("StringIndexOutOfBoundsException");          }      }  } |

**Output:**

StringIndexOutOfBoundsException

# Java try-catch block

## Problem without exception handling

Let's try to understand the problem if we don't use a try-catch block.

### Example 1

**public** **class** TryCatchExample1 {

**public** **static** **void** main(String[] args) {

**int** data=50/0; //may throw exception

        System.out.println("rest of the code");

    }

}

**Output:**

Exception in thread "main" java.lang.ArithmeticException: / by zero

## Solution by exception handling

Let's see the solution of the above problem by a java try-catch block.

### Example 2

**public** **class** TryCatchExample2 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

        }

            //handling the exception

**catch**(ArithmeticException e)

        {

            System.out.println(e);

        }

        System.out.println("rest of the code");

    }

}

**Output:**

java.lang.ArithmeticException: / by zero

rest of the code

Let's see an example to print a custom message on exception.

**public** **class** TryCatchExample5 {

**public** **static** **void** main(String[] args) {

**try**

        {

**int** data=50/0; //may throw exception

        }

             // handling the exception

**catch**(Exception e)

        {

                  // displaying the custom message

            System.out.println("Can't divided by zero");

        }

    }

}

**Output:**

Can't divided by zero

**public** **class** TryCatchExample6 {

**public** **static** **void** main(String[] args) {

**int** i=50;

**int** j=0;

**int** data;

**try**

        {

        data=i/j; //may throw exception

        }

            // handling the exception

**catch**(Exception e)

        {

             // resolving the exception in catch block

            System.out.println(i/(j+2));

        }

    }

}

**Output:**

25

# Java catch multiple exceptions

### Example 1

Let's see a simple example of java multi-catch block.

**public** **class** MultipleCatchBlock1 {

**public** **static** **void** main(String[] args) {

**try**{

**int** a[]=**new** **int**[5];

                a[5]=30/0;

               }

**catch**(ArithmeticException e)

                  {

                   System.out.println("Arithmetic Exception occurs");

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                 System.out.println("ArrayIndexOutOfBounds Exception occurs”);

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

**Output:**

Arithmetic Exception occurs

rest of the code

### Example 3

In this example, try block contains two exceptions. But at a time only one exception occurs and its corresponding catch block is invoked.

**public** **class** MultipleCatchBlock3 {

**public** **static** **void** main(String[] args) {

**try**{

**int** a[]=**new** **int**[5];

                System.out.println(a[10]);

                a[5]=30/0;

               }

**catch**(ArithmeticException e)

                  {

                   System.out.println("Arithmetic Exception occurs");

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                   System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

**Output:**

Arithmetic Exception occurs

rest of the code

### Example 3

**public** **class** MultipleCatchBlock4 {

**public** **static** **void** main(String[] args) {

**try**{

                String s=**null**;

                System.out.println(s.length());

               }

**catch**(ArithmeticException e)

                  {

 System.out.println("Arithmetic Exception occurs");

                  }

**catch**(ArrayIndexOutOfBoundsException e)

                  {

                 System.out.println("ArrayIndexOutOfBounds Exception occurs");

                  }

**catch**(Exception e)

                  {

                   System.out.println("Parent Exception occurs");

                  }

               System.out.println("rest of the code");

    }

}

**Output:**

Parent Exception occurs

rest of the code

**Let's see an example, to handle the exception without maintaining the order of exceptions (i.e. from most specific to most general).**

**class** MultipleCatchBlock5{

**public** **static** **void** main(String args[]){

**try**{

**int** a[]=**new** **int**[5];

    a[5]=30/0;

   }

**catch**(Exception e){System.out.println("common task completed");}

**catch**(ArithmeticException e){System.out.println("task1 is completed");}

**catch**(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}

   System.out.println("rest of the code...");

 }

}

**Output:**

Compile-time error

# Java Nested try block

The try block within a try block is known as nested try block in java.

### Why use nested try block

Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.

### Syntax:

....

**try**

{

    statement 1;

    statement 2;

**try**

    {

        statement 1;

        statement 2;

    }

**catch**(Exception e)

    {

    }

}

**catch**(Exception e)

{

}

....

## Java nested try example

Let's see a simple example of java nested try block.

**class** Excep6{

**public** **static** **void** main(String args[]){

**try**{

**try**{

     System.out.println("going to divide");

**int** b =39/0;

    }**catch**(ArithmeticException e){System.out.println(e);}

**try**{

**int** a[]=**new** **int**[5];

    a[5]=4;

    }**catch**(ArrayIndexOutOfBoundsException e){System.out.println(e);}

    System.out.println("other statement);

  }**catch**(Exception e){System.out.println("handeled");}

  System.out.println("normal flow..");

 }

}

# Java finally block



### Case 1

Let's see the java finally example where **exception doesn't occur**.

**class** TestFinallyBlock{

**public** **static** **void** main(String args[]){

**try**{

**int** data=25/5;

   System.out.println(data);

  }

**catch**(NullPointerException e){System.out.println(e);}

**finally**{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");

  }

}

**Output:5**

finally block is always executed

rest of the code...

**Let's see the java finally example where exception occurs and handled.**

**public** **class** TestFinallyBlock2{

**public** **static** **void** main(String args[]){

**try**{

**int** data=25/0;

   System.out.println(data);

  }

**catch**(Exception e){System.out.println(“done”);}

**finally**{System.out.println("finally block is always executed");}

  System.out.println("rest of the code...");

  }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=TestFinallyBlock2)

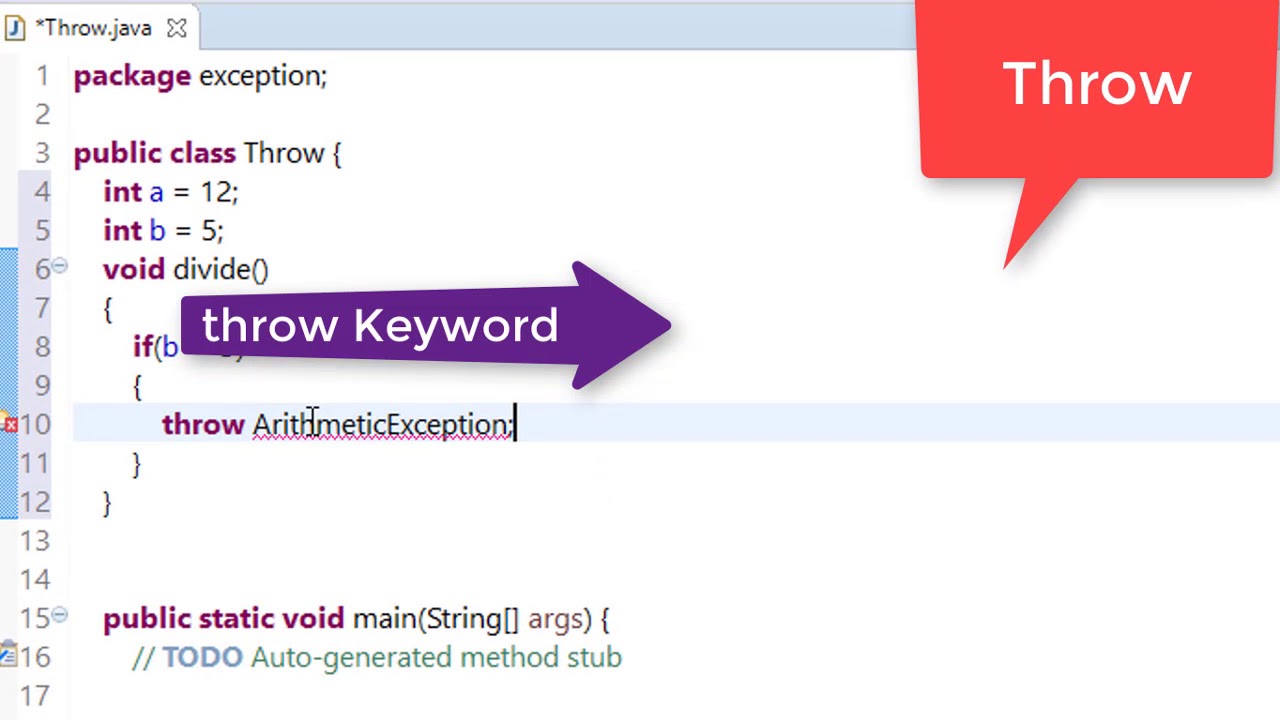
**Output:**

Exception in thread main java.lang.ArithmeticException:/ by zero

finally block is always executed

rest of the code...

**Throw Keyword**



The Java throw keyword is used to explicitly throw an exception.

We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

public class TestThrow1{

static void validate(int age){

if(age<18)

throw new ArithmeticException("not valid");

else

System.out.println("welcome to vote");

}

public static void main(String args[]){

validate(13);

System.out.println("rest of the code...");

}

}

Output:

Welcome to the Registration process!!Exception in thread "main"

java.lang.ArithmeticException: Student is not eligible for registration

at beginnersbook.com.ThrowExample.checkEligibilty(ThrowExample.java:9)

at beginnersbook.com.ThrowExample.main(ThrowExample.java:18)

**Throws Keyword**

Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

import java.io.IOException;

class Testthrows1{

void m()throws IOException{

throw new IOException("device error");//checked exception

}

void n()throws IOException{

m();

}

void p(){

try{

n();

}catch(Exception e){System.out.println("exception handled");}

}

public static void main(String args[]){

Testthrows1 obj=new Testthrows1();

obj.p();

System.out.println("normal flow...");

}

}

Output:

exception handled

normal flow...

# Difference between throw and throws in Java

Both throw and throws are the concepts of exception handing in which throw is used to explicitly throw an exception from a method or any block of code while throws are used in the signature of the method to indicate that this method might throw one of the listed type exceptions.

The following are the important differences between throw and throws.

| **Sr. No.** | **Key** | **throw** | **throws** |
| --- | --- | --- | --- |
| 1 | Definition | Throw is a keyword which is used to throw an exception explicitly in the program inside a function or inside a block of code. | Throws is a keyword used in the method signature used to declare an exception which might get thrown by the function while executing the code. |

## Example of throw vs throws

**JavaTester.java**

public class JavaTester{

   public void checkAge(int age){

      if(age<18)

         throw new ArithmeticException("Not Eligible for voting");

      else

         System.out.println("Eligible for voting");

   }

   public static void main(String args[]){

      JavaTester obj = new JavaTester();

      obj.checkAge(13);

      System.out.println("End Of Program");

   }

}

## Output

Exception in thread "main" java.lang.ArithmeticException:

Not Eligible for voting

at JavaTester.checkAge(JavaTester.java:4)

at JavaTester.main(JavaTester.java:10)

## Example

**JavaTester.java**

public class JavaTester{

   public int division(int a, int b) throws ArithmeticException{

      int t = a/b;

      return t;

   }

   public static void main(String args[]){

      JavaTester obj = new JavaTester();

      try{

         System.out.println(obj.division(15,0));

      }

      catch(ArithmeticException e){

         System.out.println("You shouldn't divide number by zero");

      }

   }

}

## Output

You shouldn't divide number by zero